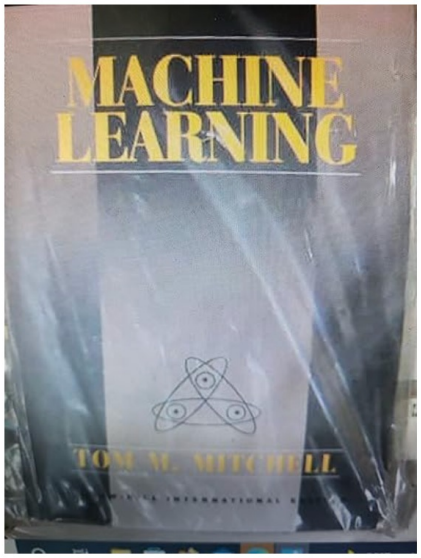
Basic Python Programming

Reference books



Roll over image to zoom in

Machine Learning (McGraw-Hill International Editions Computer Science Series) Paperback – January 1, 1997

by Tom M. Mitchell (Author)

4.3 ★★★★½ 115 ratings

See all formats and editions

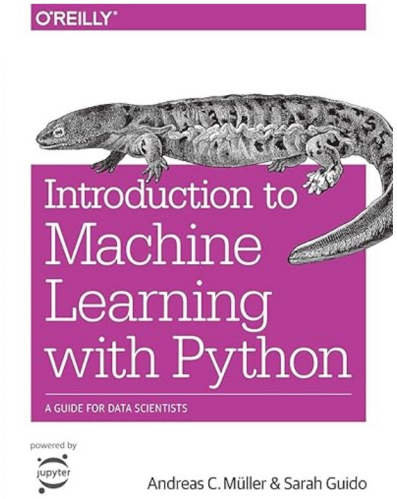
This book covers the field of machine learning, which is the study of algorithms that allow computer programs to automatically improve through experience. The book is intended to support upper level undergraduate and introductory level graduate courses in machine learning.

Report an issue with this product or seller

Print length	Language	Publisher	Publication date	Dimensions
414 pages	English	McGraw-Hill	January 1, 1997	6.6 x 1.3 x 9.5 inches

Reference books

‹ Back to results



Roll over image to zoom in

Introduction to Machine Learning with Python: A Guide for Data Scientists 1st Edition

by Andreas Müller (Author), Sarah Guido (Author)

4.6 ★★★★½ 542 ratings

Part of: Learning Python (7 books)

See all formats and editions

Machine learning has become an integral part of many commercial applications and research projects, but this field is not exclusive to large companies with extensive research teams. If you use Python, even as a beginner, this book will teach you practical ways to build your own machine learning solutions. With all the data available today, machine learning applications are limited only by your imagination.

You'll learn the steps necessary to create a successful machine-learning application with Python and the scikit-learn library. Authors Andreas Müller and Sarah Guido focus on the practical aspects of using machine learning algorithms, rather than the math behind them. Familiarity with the NumPy and matplotlib libraries will help you get even more from this book.

With this book, you'll learn:

- Fundamental concepts and applications of machine learning
- Advantages and shortcomings of widely used machine learning algorithms

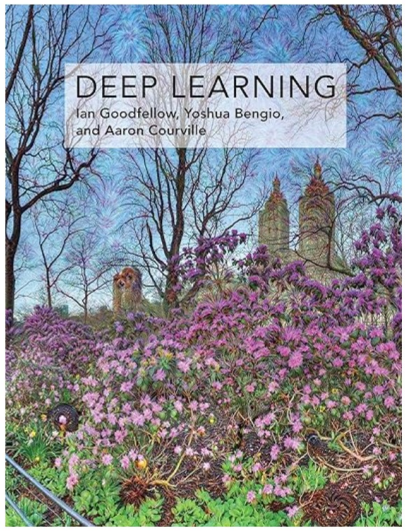
Read more

Report an issue with this product or seller

ISBN-10	ISBN-13	Edition	Publisher	Publication date

Reference books

‹ Back to results



Roll over image to zoom in

Read sample

Deep Learning (Adaptive Computation and Machine Learning series)

by Ian Goodfellow (Author), Yoshua Bengio (Author), Aaron Courville (Author)

4.3 ★★★★☆ 2,163 ratings

See all formats and editions

An introduction to a broad range of topics in deep learning, covering mathematical and conceptual background, deep learning techniques used in industry, and research perspectives.
"Written by three experts in the field, *Deep Learning* is the only comprehensive book on the subject."
—Elon Musk, cochair of OpenAI; cofounder and CEO of Tesla and SpaceX

Deep learning is a form of machine learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts. Because the computer gathers knowledge from experience, there is no need for a human computer operator to formally specify all the knowledge that the computer needs. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones; a graph of these hierarchies would be many layers deep. This book introduces a broad range of topics in deep learning.

The text offers mathematical and conceptual background, covering relevant concepts in linear algebra, probability theory and information theory, numerical computation, and machine learning. It describes deep learning techniques used by practitioners in industry, including deep feedforward networks, regularization, optimization algorithms,

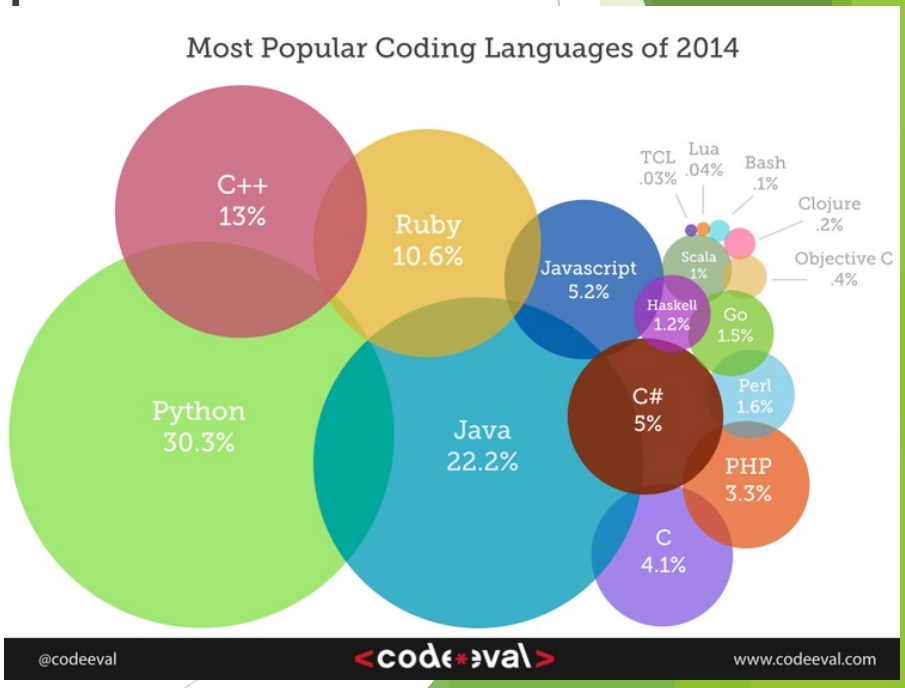
Read more

Report an issue with this product or seller

ISBN-10	ISBN-13	Publisher	Publication date	Language

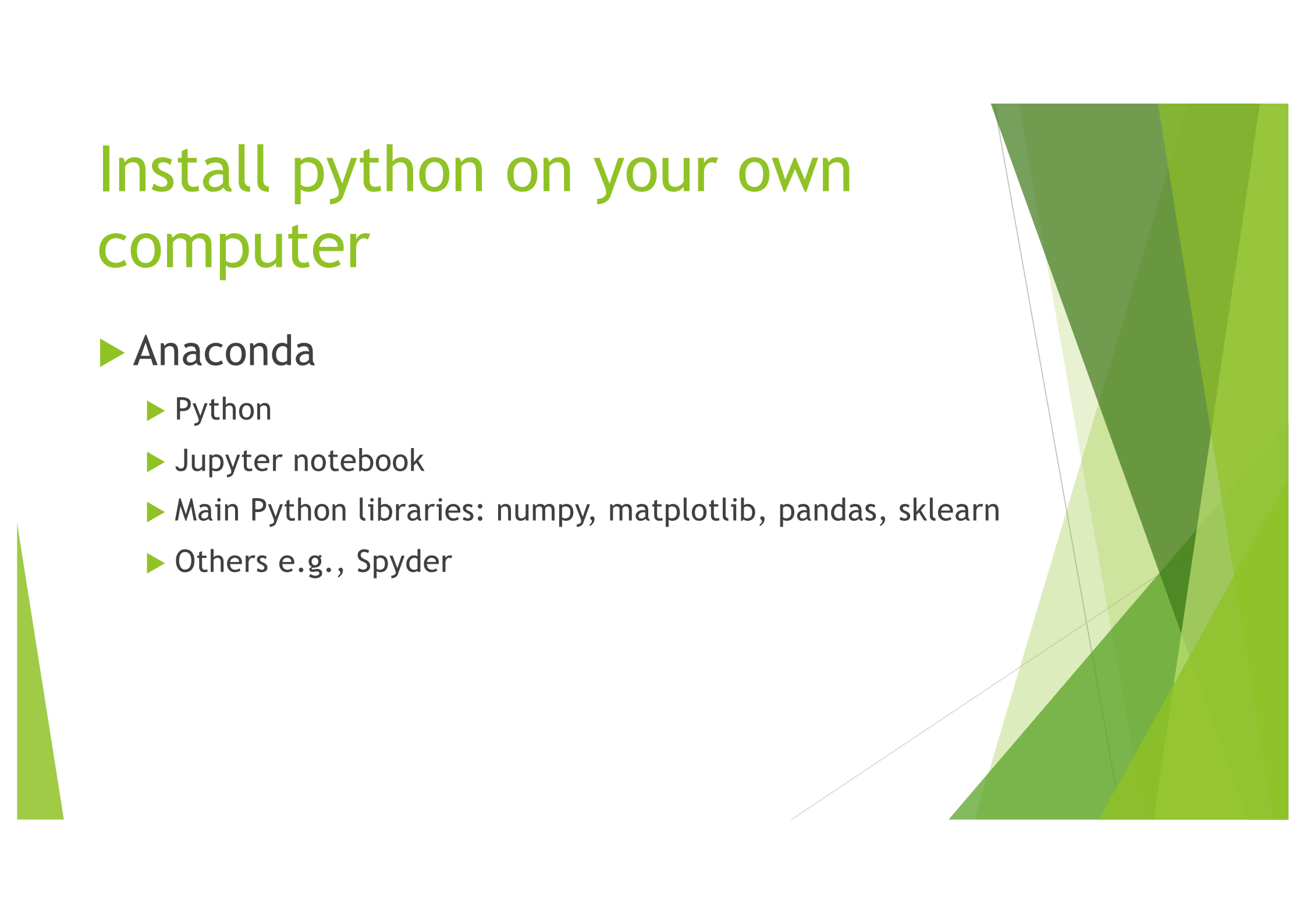
Why Python?

- Python is the most popular language
- Python has many libraries
- Good documentation, good support
- Python is
  - Concise/precise
  - Easy for beginners



Install python on your own computer

- Anaconda
  - Python
  - Jupyter notebook
  - Main Python libraries: numpy, matplotlib, pandas, sklearn
  - Others e.g., Spyder



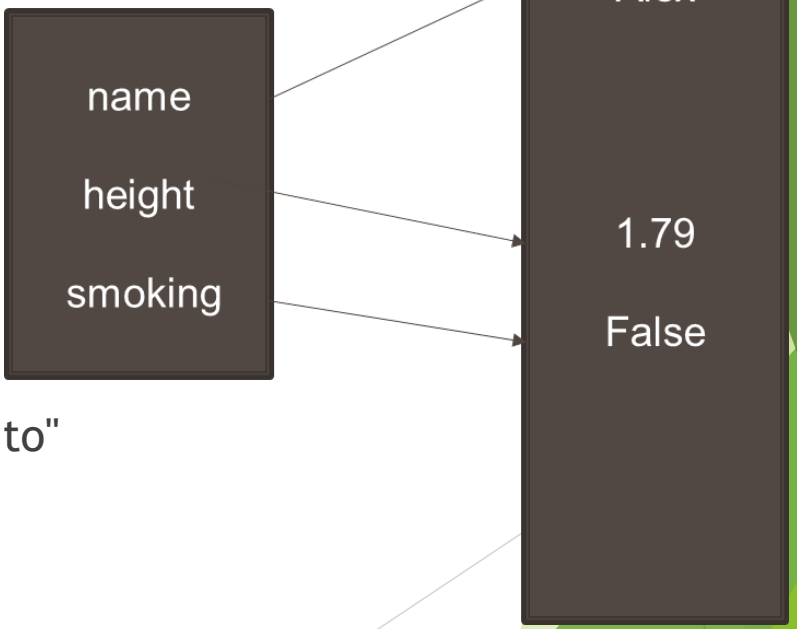
Using Jupyter notebook

- Browse the directories
- Create a new folder and rename it (if you have not done so using the operating system)
- Create a new file and give it a good name
- Write your first Python program
- Run the first Python program
  - Ctrl+Enter / Shift+Enter

Variables

- Examples

```
name = "Alex"
age = 20
height = 1.79
smoking = False

course = ”AIML231"
required_course = True
size = 175
```



- Assignment statement
  - = is not “equals”, is "assign to", "bind name to"
  - create a reference (address)

- Question: Why variables?

Variable has a data type

- Name, value, data type
- What data type?
  - Boolean: **bool**, **True** or **False**.
  - Integer: **int**.
  - Real numbers: **float**.
  - String: **str**
  - …
  - **list, dict, tuple, Series, DataFrame**
- Why data type?
  - Size, how they are represented/stored,
  - Each type allow certain operations
  - Error checking

Using variables, operators, expressions

```
course = "AIML231"
print("Welcome to " + course)
```

- Operators
  - String concatenation: **+**
  - Arithmetic operators: **+, - , *, / , // (integer division), % (remainder), ** (to the power of)**

```
x = 138
y = 7282488
z = 48 * x + y/2 + x * y + 32 * x **2
print(z)
```

- Expressions:
  - An expression is a combination of values, variables, operators
  - (Expressions can have calls to functions, later in the course).
  - Arithmetic expressions,
  - (Boolean expressions, later)

Input & output

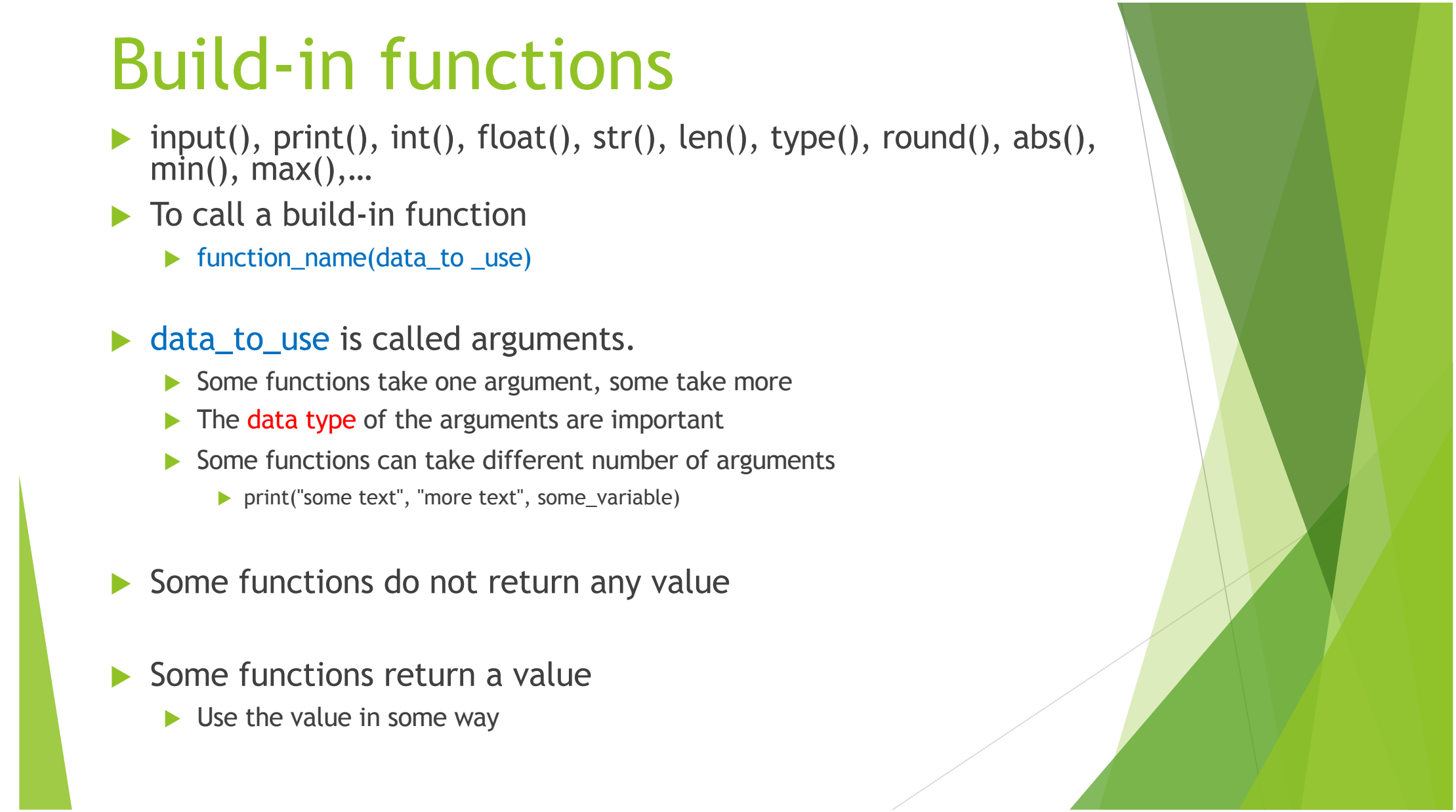
- Write a birthday invitation

```
name = input("Enter a name here: ")
print("Dear " + name)
print("You are invited to my birthday
party")
```

- **input** is a built-in function
- "enter a name here" is a string (**str**), the prompt to user
- this function returns a string (**str**)

Build-in functions

- input(), print(), int(), float(), str(), len(), type(), round(), abs(), min(), max(),…
- To call a build-in function
  - function_name(data_to _use)
- data_to_use is called arguments.
  - Some functions take one argument, some take more
  - The data type of the arguments are important
  - Some functions can take different number of arguments
    - print("some text", "more text", some_variable)
- Some functions do not return any value
- Some functions return a value
  - Use the value in some way



Example

```
round(4.29)

abs(-4.5)

int(458.645889)
```

Function, arguments, returned value

Define your own function

- Functions are defined with three components:

- **The header**, which includes the **def** keyword, the name of the function, and any parameters the function requires.
  - Parameter: data required, optional

- Comment: optional, strongly recommended,
  - triple quotes for method comments
  - will be printed when using help()

- The body: must be indented, 4 spaces recommended (use tab key)

```
def hello():
      """Prints 'Hello Everyone!' to the console."""
      print("Hello Everyone! ")
```

Function with parameters

- Parameters are the variables for this function

```
def welcome(name, course, num):
    """Print a welcome message.
       It has three parameters:
       name as a str, course as a str and num as an int
    """
    print("hello " + name)
    print("Welcome to " + course)
    print("There are " + str(num) +" students in our class")
```

- When you call a function, specify the data_to_use (arguments)
- Arguments are passed to the parameters

```
welcome("Alex", ”AIML231", 180)
welcome("Bob", ”AIML231", 420)
welcome("Sara", ”AIML232", 100)
welcome(”AIML231", "Alex", 180)
welcome(”AIML231", 180, "Alex")
```

Keyword arguments

- Specify the parameter name of the arguments, then the order is not important

```
welcome(name="Alex", num=180, course="COMP132")
```

- Positional arguments must be before keyword arguments

```
welcome("Liz", num=120, course="WRIT151")
```

Parameter with default values

- Define a function using parameters with default values

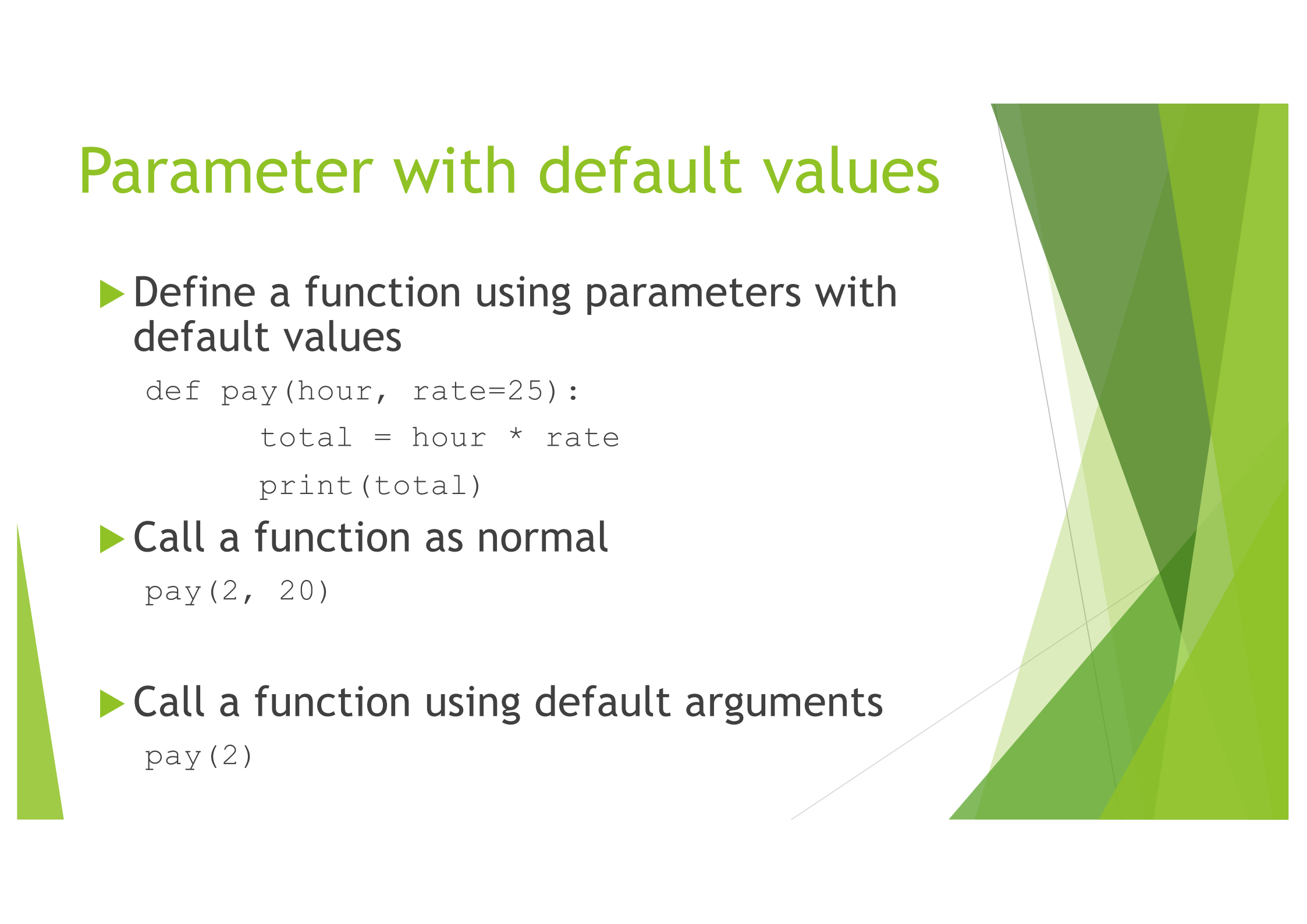
```
def pay(hour, rate=25):
        total = hour * rate
        print(total)
```

- Call a function as normal

```
pay(2, 20)
```

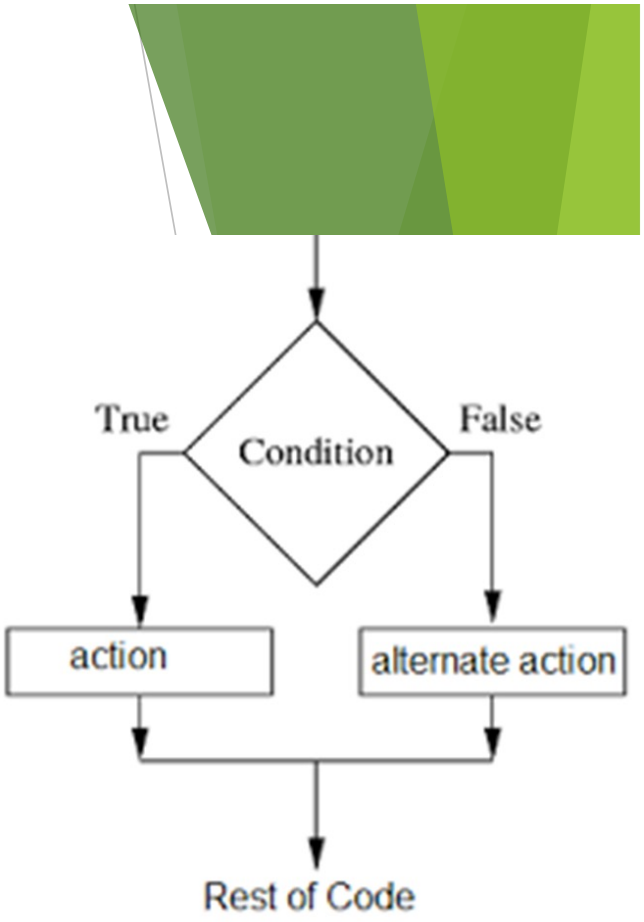
- Call a function using default arguments

```
pay(2)
```



If else

```
num = float(input("enter a number: "))
if num >= 0:
      print("Positive or Zero")
else:
      print("Negative number")
```



- Template

if some_condition :
*   do_something*
else:
*   do something_else*

Example

```
def greater_less_equal_5(answer):
    if answer > 5:
        return 1
    elif answer < 5:
        return -1
    else:
        return 0

print(greater_less_equal_5(4))
print(greater_less_equal_5(5))
print(greater_less_equal_5(6))
```

Exercise

- Define a function that adds 15% tax to an amount
- Define a function that adds 10% tip to an amount

- Call the functions to work out how much to pay for a giving amount. If the taxed amount is less than $100, no tip applies.

Libraries, modules

- Python have many useful functions
- Only a very small amount of them are built-in
- Most functions are organised in libraries of code called Modules.
- A module is simply a file containing Python definitions, functions, and statements.

- Must import the module
- Call a function: specify module name and then function name

Two common ways to import a module

- Can import the entire module and its *namespace*

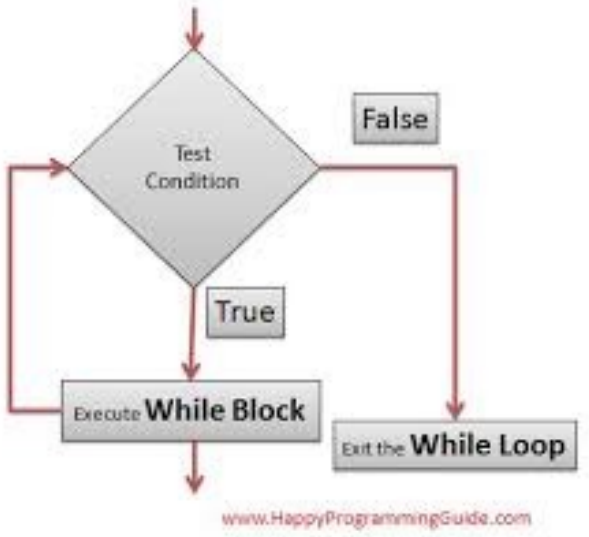
```
import math
print(math.pi)                                  #
An imported data value
print(math.sqrt(23.456))                  # An
imported function
```

- Or we can import selected data/functions into current namespace

```
from math import pi, sqrt
print(pi)
print(sqrt(23.456))
```

Using loops

LOOPS REPEAT ACTIONS...
SO YOU DON'T HAVE TO

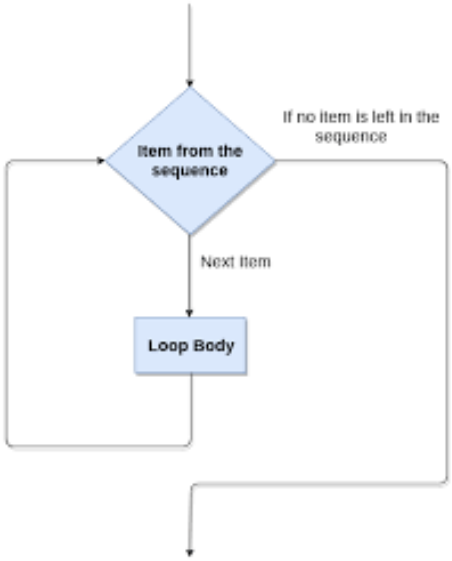


- Loop

  while *the condition is true:*

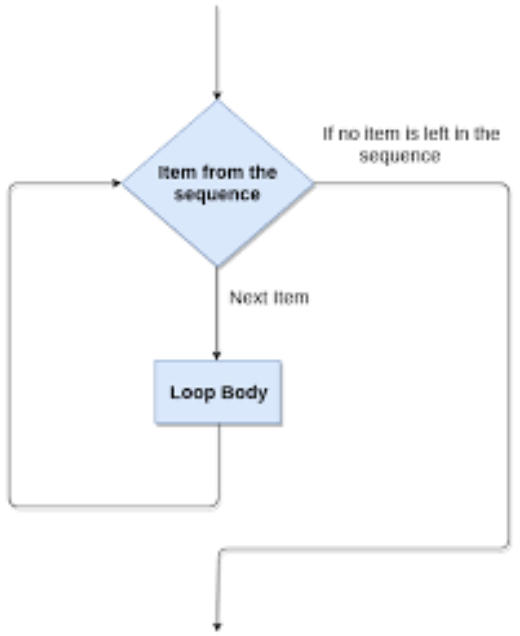
  loop in the body

- "for" loop
  - Executing a statement a given number of times
  - Using "for" loop on Strings



Typical for loop

```
for x in range(10):
    print(x)
    print("hello")
```



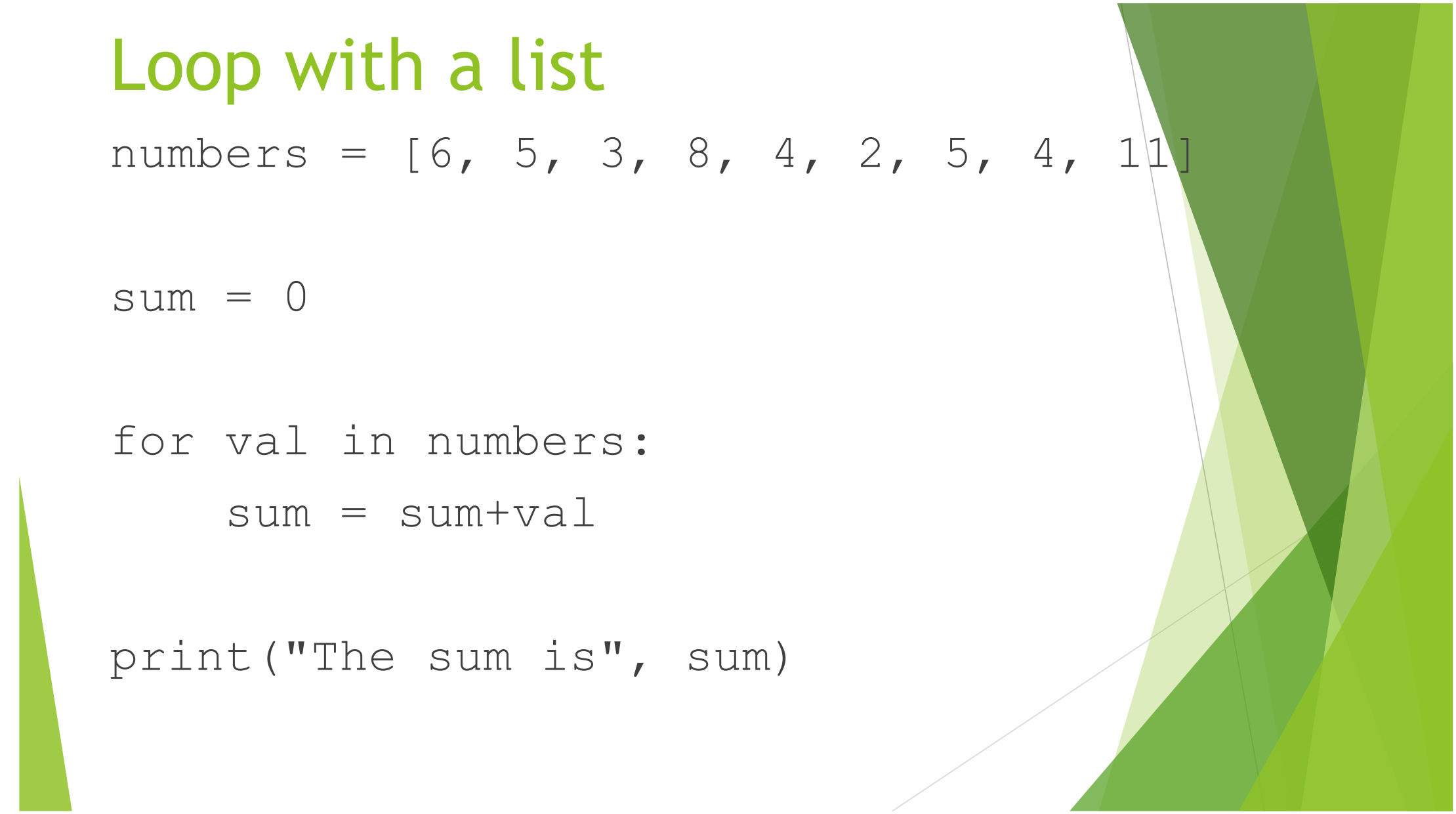
range() is a built-in function.

range(10) returns a special object that when you iterate over it, gives you the numbers 0 through 9

in is a built-in operator

Loop with a list

```
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

sum = 0

for val in numbers:
    sum = sum+val

print("The sum is", sum)
```



Break out of a loop

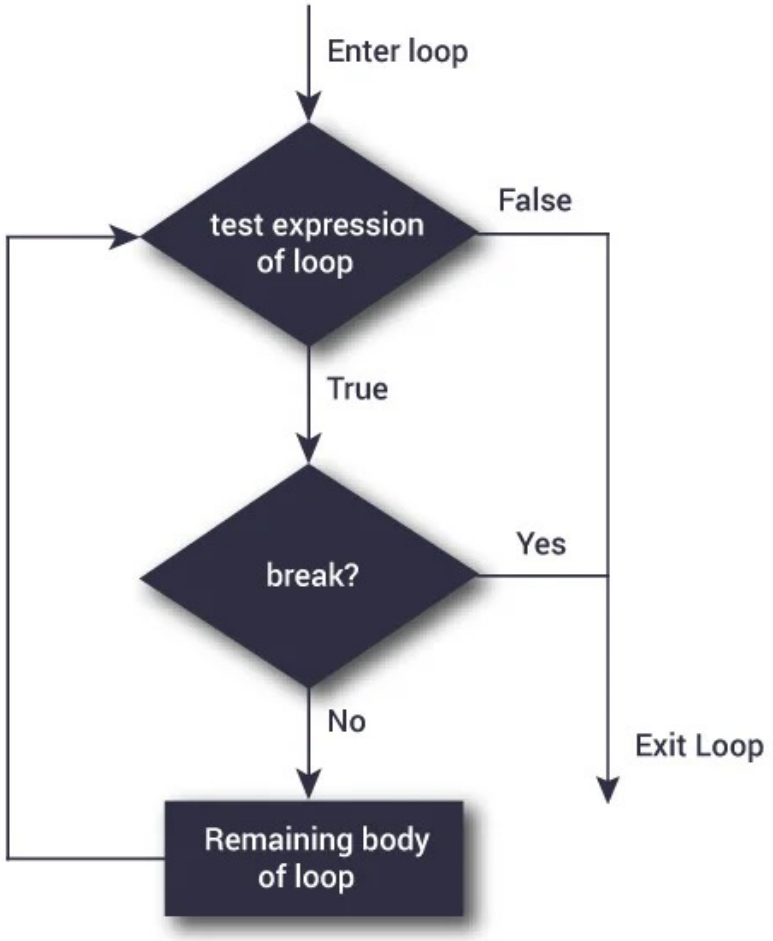
- To exit a loop

- Typically

for var in sequence:

    if (condition):

        break




```
for letter in 'Python':
    if letter == 'h':
        break
    print('Current Letter :' + letter)
```

str data type: how to get a string?

- Built-in data type
- Create a **str** object, can use " ", ' ' , ''' '''

string_name = "any words you like"

```
s = "python language"
```

- str() converts other data type to **str** data type

```
num_string = str(44848.45)
```

- Call a function that returns a **str**

```
course = input("enter the course code here: ")
```

  - Many functions return a string, e.g., read a file line by line

String and its index

- You can think of a Python string as a list of characters.
- The string "PYTHON" has six characters,
- numbered 0 to 5, as shown below:

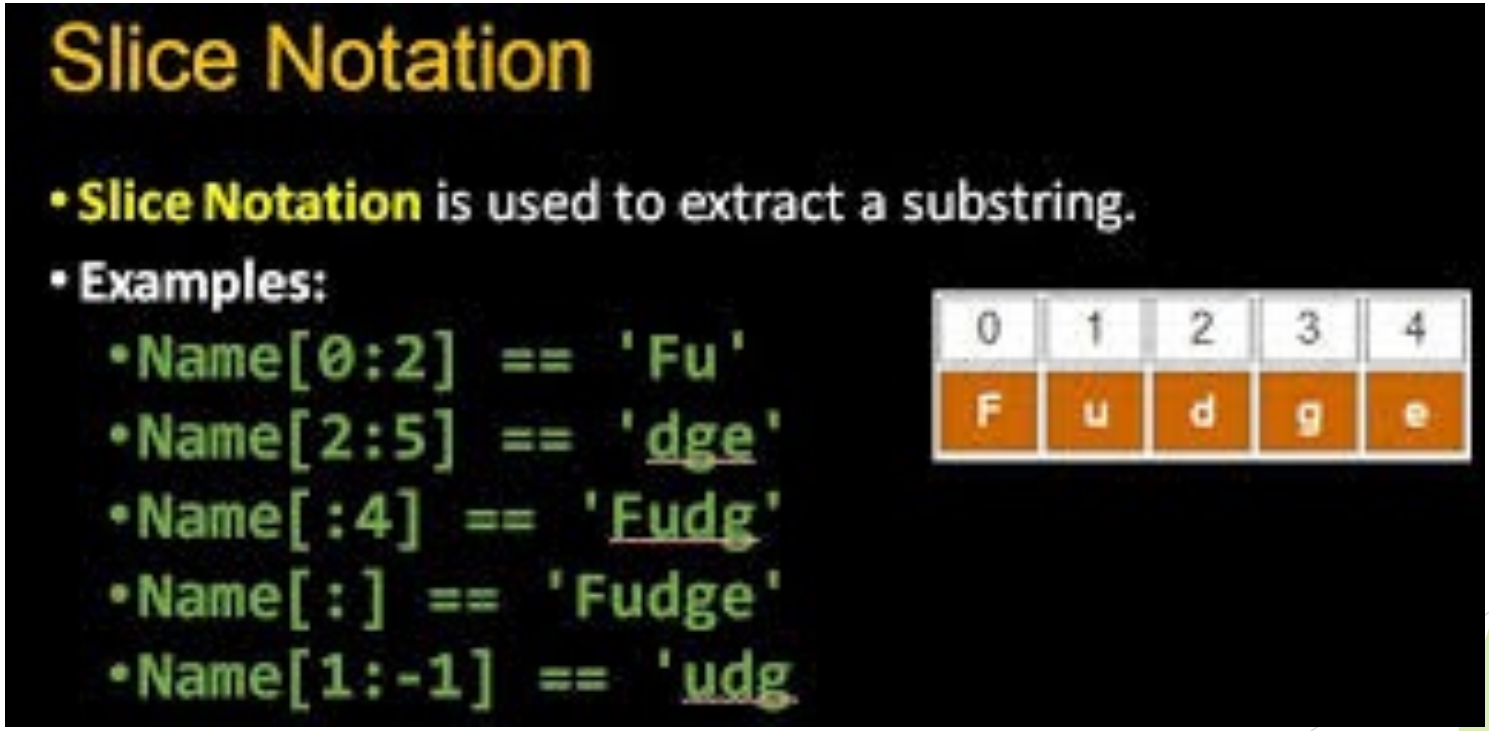
```
| P | Y | T | H | O | N |
0   1   2   3   4   5
```

- So if you wanted "Y", you could just type "PYTHON"[1]
  - (always start counting from 0!)

```
one_letter = "MONTY"[2]
print(one_letter)

m = "keep calm and carry on"
print(m[3])
```

String slicing

Name = ‘Fudge’



String methods

- s is a string object, e.g. s ="PYTHON is cool!"
- s.split()

- s.lower()
- s.upper()

- s.startswith("hi") # can be used as the Boolean condition in if statement
- s.endswith("!") # can be used as the Boolean condition in if statement

- s.find("TH")
- s.count("o")

- s.replace(), s.format(), s.isnumeric(), ….

- Documentation: https://docs.python.org/3/library/stdtypes.html#textseq
- A good one with examples at
  - https://www.programiz.com/python-programming/methods/string

How to get a list

- How to create a list

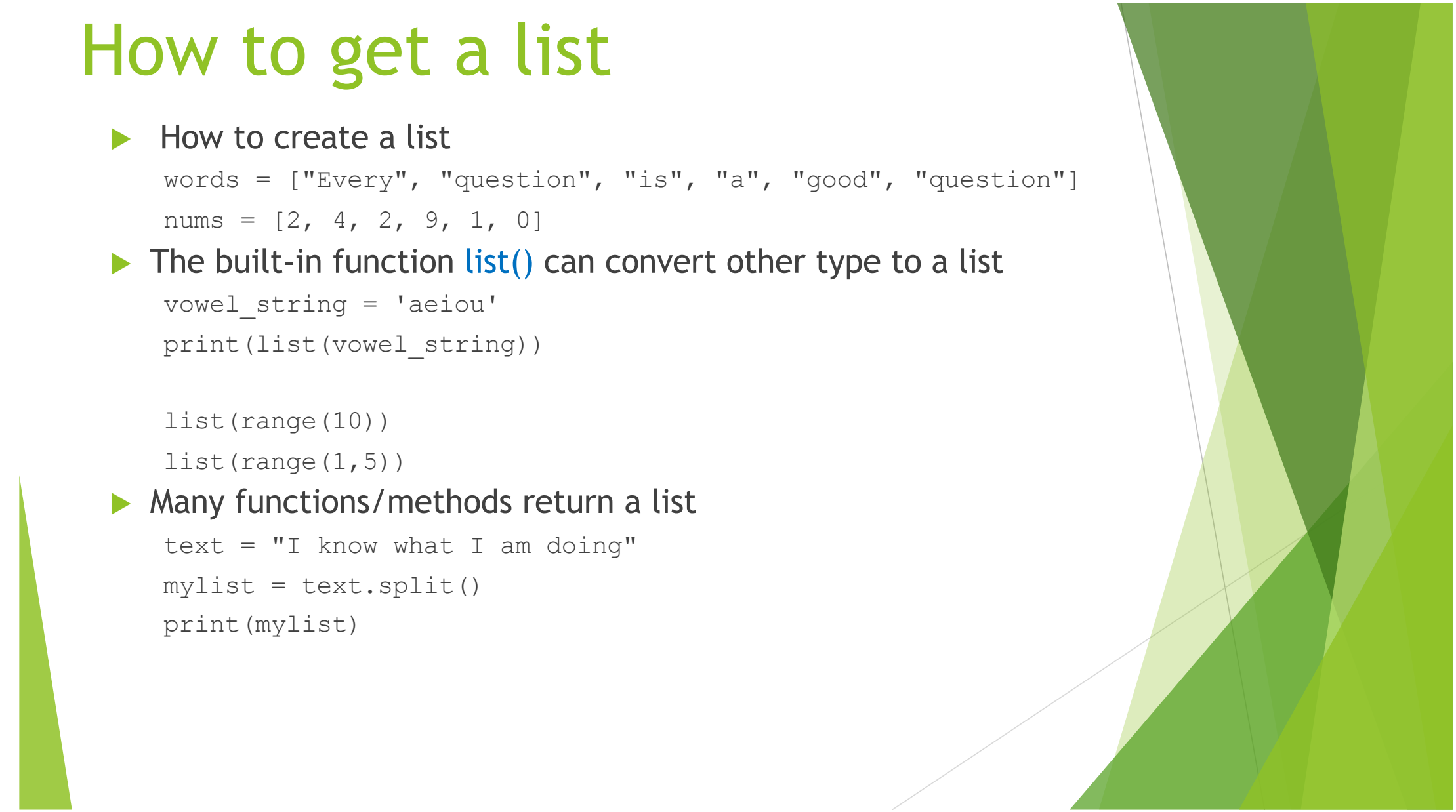
```
words = ["Every", "question", "is", "a", "good", "question"]
nums = [2, 4, 2, 9, 1, 0]
```

- The built-in function list() can convert other type to a list

```
vowel_string = 'aeiou'
print(list(vowel_string))

list(range(10))
list(range(1,5))
```

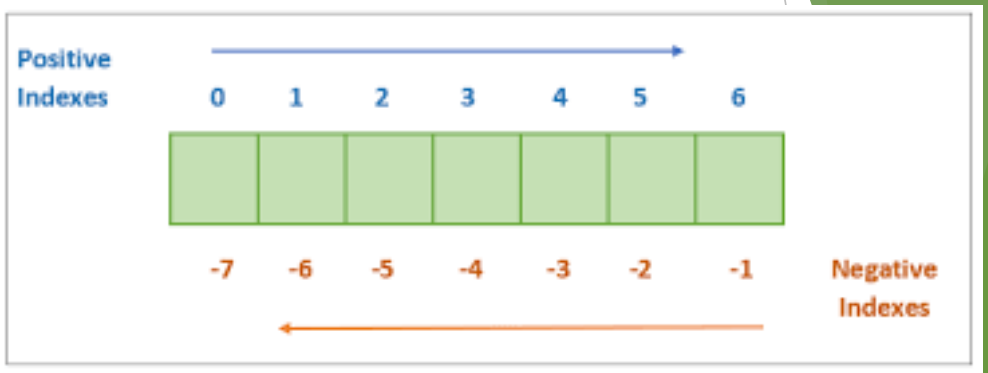
- Many functions/methods return a list

```
text = "I know what I am doing"
mylist = text.split()
print(mylist)
```



list is an indexed data structure

- Each element has an index
  - Starting from zero



- To access an element
  - *List_name[index_number]*

```
names = ["Liz", "Sam", "Alex", "Tom"]
names[0]
```

- To change an element

```
names[1] = "Bob"
```

(use index can change the list item)

- To get a sublist

```
names[1:3]
```

  - *list_name[start_at : end_before: stride]*

List methods

- Methods that can be called from a list object using dot notation
- Most methods can change the list

  e.g., data = ["a", "list", "of", "data"]

  data.reverse()

  data.sort()

  data.insert()

  data.remove()

  data.append()

  data.count()

Many more

- Documentation: https://docs.python.org/3/library/index.html
- Learn with examples: https://www.programiz.com/python-programming/methods/list
- https://www.w3schools.com/python/python_ref_list.asp

List operators

- Addition operator +:

```
odd = [1, 3, 5]
print(odd + [9, 7, 5])
print(odd)
```

- Multiplication operator *:

```
odd = [1, 3, 5]
print(odd * 3)
```

- Del operator:

```
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']

# delete one item
del my_list[2]

print(my_list)

del my_list
```

tuple

- A tuple is a type of sequence that is very similar to list, but the length is fixed and it is immutable
- If you have a sequence that does not change, the right data structure should be tuple

- Create a tuple using ( )
  - fruit=("apple", "banana", "pear") ok without the ( )
  - type(fruit)
  - Tuple with a single element n =(3,)
- Convert other type to a tuple
  - tuple([1,5,7,3])
  - tuple("aeuio")
- Many functions/methods return a tuple (when a method returns many values, it is easy to pack them into a tuple)
  - e.g., when you use Scipy to find the peaks, they are returned as a tuple, you need to use result[0] to get them.

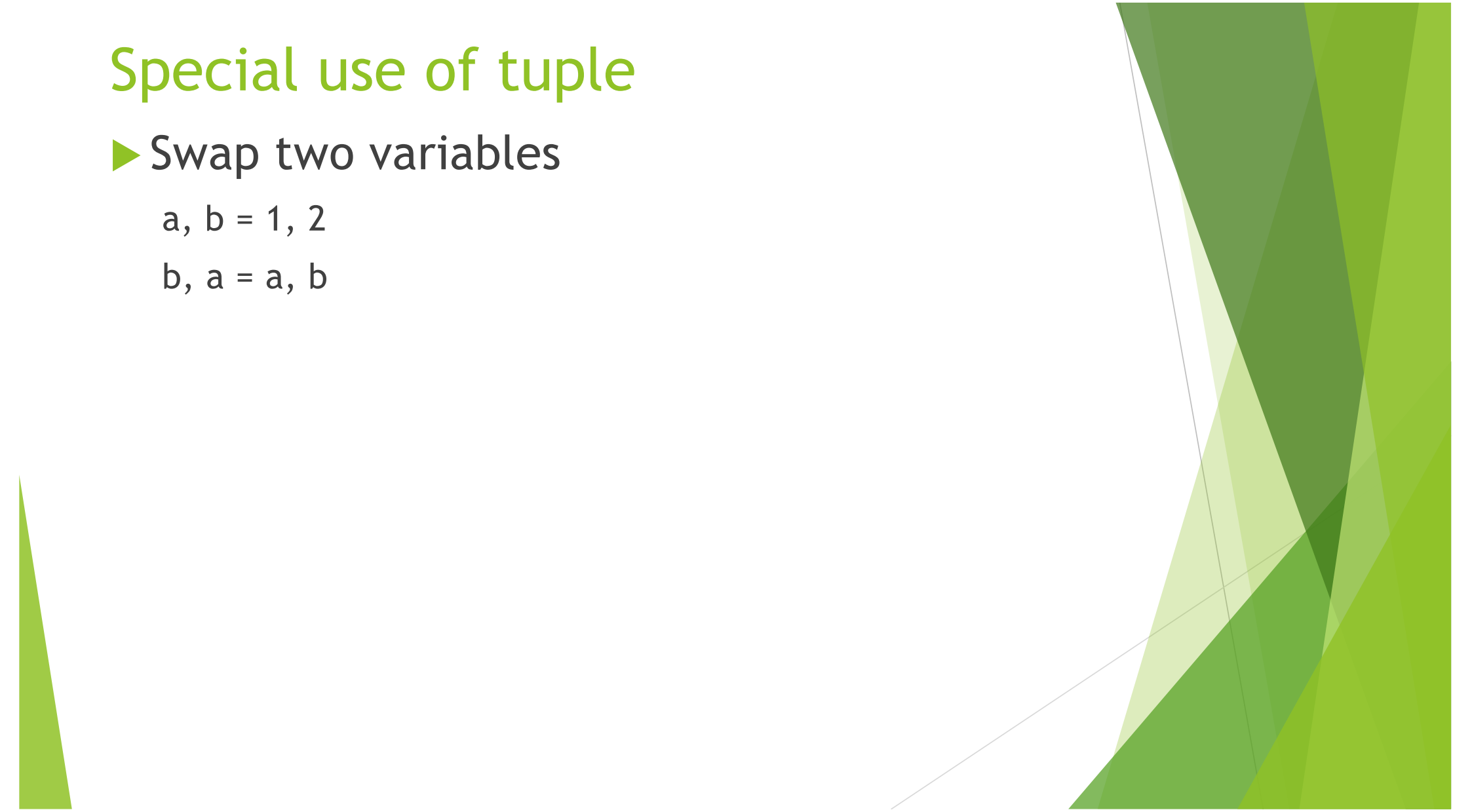
A function that returns a tuple

```
def test():
    return 'abc', 100, [0, 1, 2]

a, b, c = test()

print(a)
# abc

print(b)
# 100

print(c)
# [0, 1, 2]
```

Special use of tuple

- Swap two variables

```
a, b = 1, 2
b, a = a, b
```



Understand Python dictionary

- Dictionary is a data structure that has a key and a value.
  - Each key must be unique in the dictionary

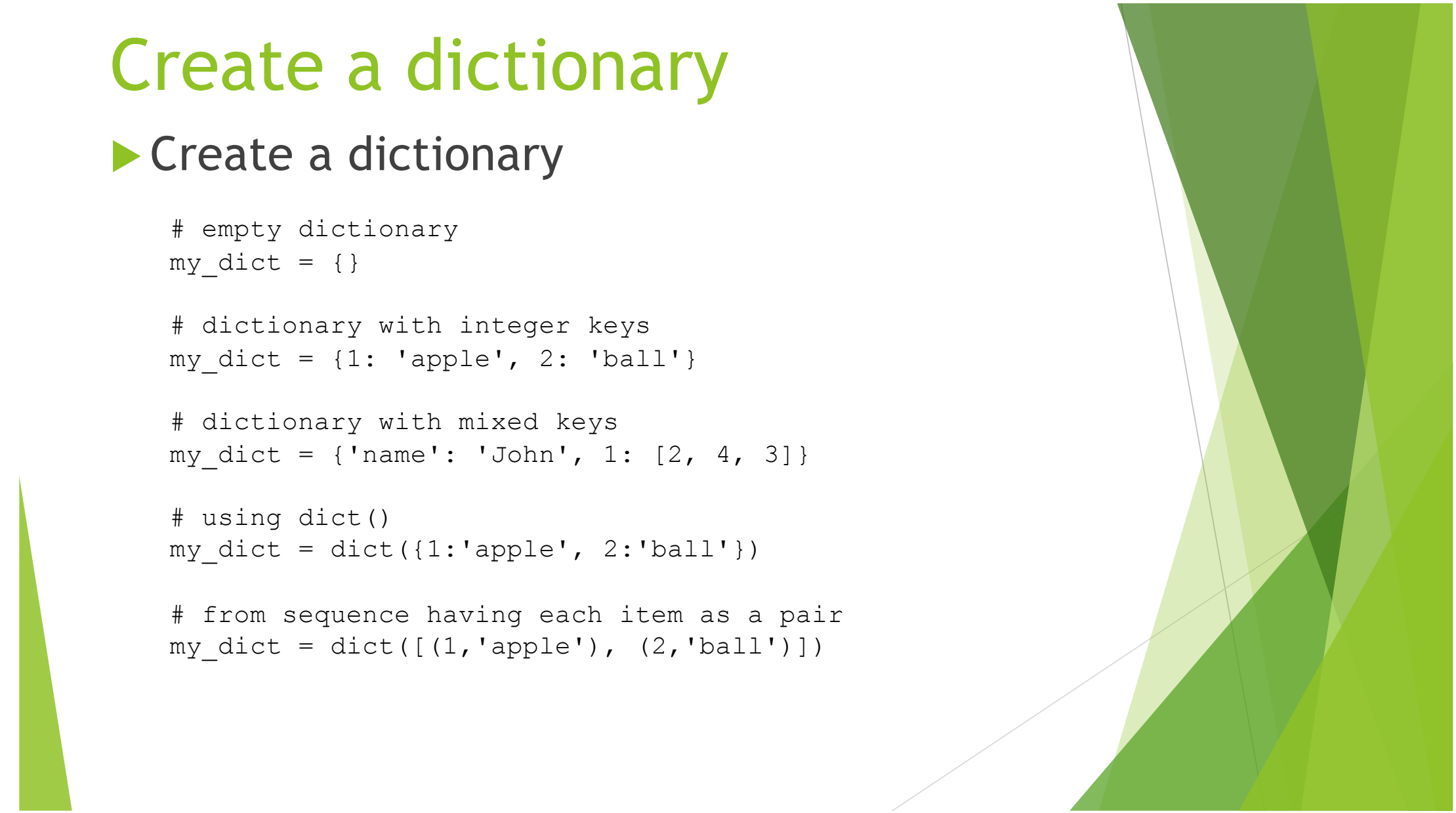
Key (Student ID Number)	Value (Persons Name)
127323	'Mike'
187428	'Tomoki'
493209	'Raoul'

- Example dictionaries
  - A phone book:
    - Key – The phone number
    - Value – The persons' name
  - A physical dictionary (hence where the name of this data structure comes from)
    - Key – The word
    - Value – The description of the word (i.e., the definition).
  - A Student record at a university
    - Key – the student ID
    - Value – The student's name

Create a dictionary

- Create a dictionary

```
# empty dictionary
my_dict = {}

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()
my_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```



Handle all elements of a dictionary

- Use a for loop to process all key-value pairs in a dictionary

```
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
for k,v in squares.items():
    print(k)
    print(v)
```